Next Generation Grid: Integrating Parallel and Distributed Computing Runtimes from Cloud to Edge Applications

The 15th IEEE International Symposium on Parallel and Distributed Processing with Applications (IEEE ISPA 2017) Guangzhou, China, December 12-15, 2017

http://trust.gzhu.edu.cn/conference/ISPA2017/

Geoffrey Fox, December 13, 2017

Department of Intelligent Systems Engineering

gcf@indiana.edu, http://www.dsc.soic.indiana.edu/, http://spidal.org/

Wòrk with Judy Qiu, Shantenu Jha, Supun Kamburugamuve, Kannan Govindarajan, Pulasthi Wickramasinghe

Abstract

- We look again at Big Data Programming environments such as Hadoop, Spark, Flink, Heron, Pregel; HPC concepts such as MPI and Asynchronous Many-Task runtimes and Cloud/Grid/Edge ideas such as event-driven computing, serverless computing, workflow and Services.
- These cross many research communities including distributed systems, databases, cyberphysical systems and parallel computing which sometimes have inconsistent worldviews.
- There are many common capabilities across these systems which are often implemented differently in each packaged environment. For example, communication can be bulk synchronous processing or data flow; scheduling can be dynamic or static; state and fault-tolerance can have different models; execution and data can be streaming or batch, distributed or local.
- We suggest that one can usefully build a toolkit (called Twister2 by us) that supports these different choices and allows fruitful customization for each application area. We illustrate the design of Twister2 by several point studies.

Predictions/Assumptions

- **Supercomputers** will be essential for large simulations and will run other applications
- **HPC Clouds** or **Next-Generation Commodity Systems** will be a dominant force
  - Merge **Cloud HPC** and (support of) **Edge** computing
  - Federated Clouds running in multiple giant datacenters offering all types of computing
  - Distributed data sources associated with device and Fog processing resources
  - **Server-hidden computing** and **Function as a Service FaaS** for user pleasure "No server is easier to manage than no server"
  - **Support a distributed event-driven serverless dataflow computing model covering batch and streaming data as HPC-FaaS**
  - Needing parallel and distributed (Grid) computing ideas
  - Span **Pleasingly Parallel** to **Data management** to **Global Machine Learning**

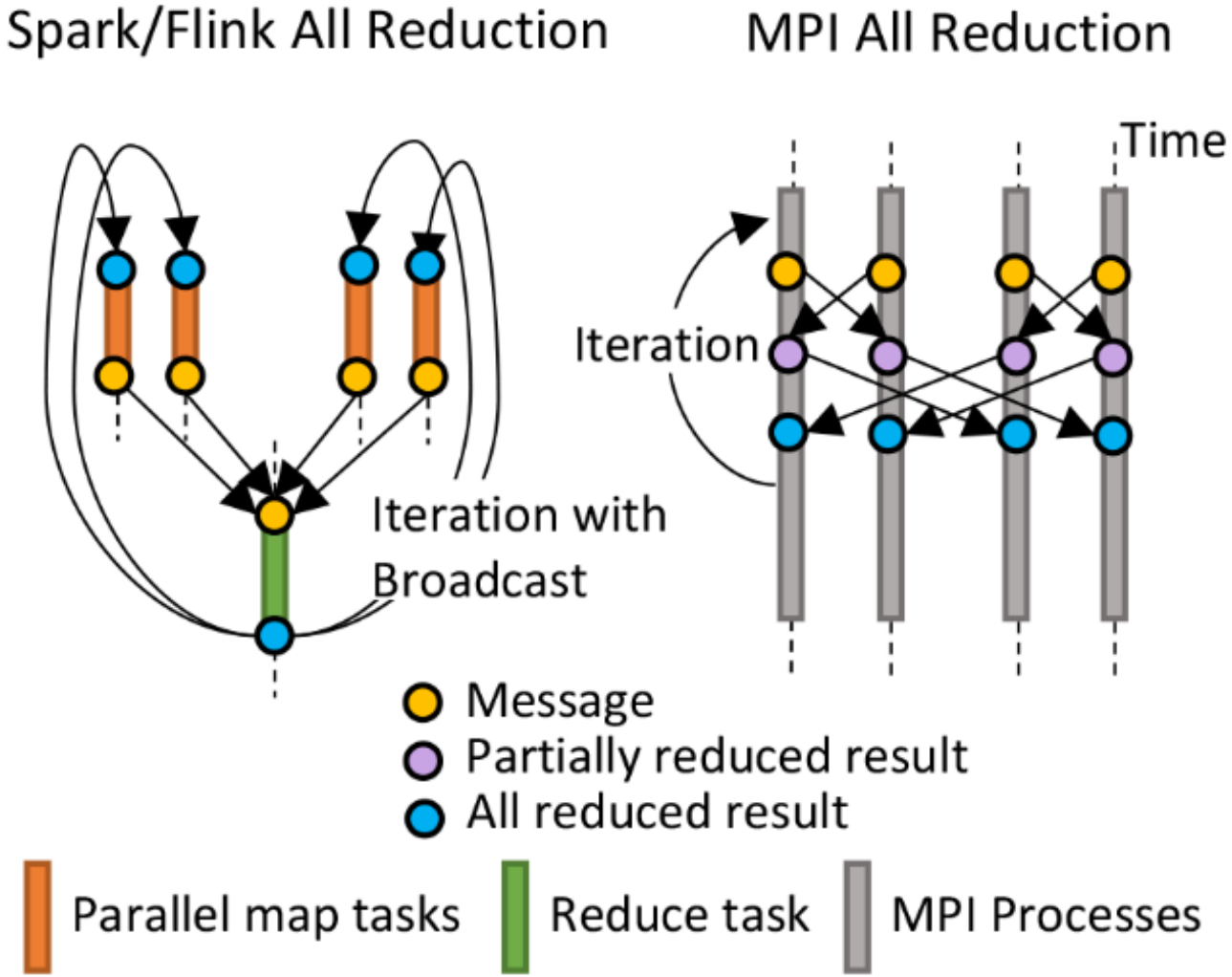
Background Remarks

- **Use of public clouds increasing rapidly**
  - Clouds becoming diverse with subsystems containing GPU's, FPGA's, high performance networks, storage, memory …
- **Rich software stacks**:
  - HPC (High Performance Computing) for Parallel Computing **less used than(?)**
  - Apache for Big Data Software Stack ABDS including center and edge computing (streaming)
- Surely **Big Data** requires **High Performance Computing**?
- **Service-oriented Systems, Internet of Things** and **Edge Computing** growing in importance
- A lot of **confusion** coming from **different communities** (database, distributed, parallel computing, machine learning, computational/data science) investigating similar ideas with little knowledge exchange and mixed up (unclear) requirements

Requirements

- On general principles **parallel and distributed computing** have different requirements even if sometimes similar functionalities
  - Apache stack ABDS typically uses distributed computing concepts
  - For example, Reduce operation is different in MPI (Harp) and Spark
- Large scale simulation requirements are well understood
- Big Data requirements are not agreed but there are a few key use types
  1) **Pleasingly parallel** processing (including **local machine learning LML**) as of different tweets from different users with perhaps MapReduce style of statistics and visualizations; possibly Streaming
  2) **Database model** with queries again supported by MapReduce for horizontal scaling
  3) **Global Machine Learning GML** with single job using multiple nodes as classic parallel computing
  4) **Deep Learning** certainly needs HPC – possibly only multiple small systems
- Current workloads stress 1) and 2) and are suited to current clouds and to ABDS (with no HPC)
  - This explains why Spark with poor GML performance is so successful and why it can ignore MPI even though MPI uses best technology for parallel computing

HPC Runtime versus ABDS distributed Computing Model on Data Analytics



Hadoop writes to disk and is **slowest**; **Spark** and **Flink** spawn many processes and do not support AllReduce directly;
MPI does in-place combined reduce/broadcast and is **fastest**

Need Polymorphic Reduction capability choosing best implementation

Use HPC architecture with
Mutable model
Immutable data

Use Case Analysis

- Very short as described in previous talks and papers
- Started with NIST collection of 51 use cases
- “Version 2” https://bigdatawg.nist.gov/V2_output_docs.php just released August 2017
- 64 Features of Data and Model for large scale big data or simulation use cases

NIST Big Data Program

Welcome to NIST Big Data Public Working Group (NBD-PWG)! Search GO

Home
NBD-WG/Subgroups
Charter
Co-Chairs
Guidelines
All WG Meeting

Documents
Version 2 Drafts
Version 1 Final Docs
Docs Repository
Use Cases Listing
Upload Document

Registration
New User
Update Profile

Points of Contact
Wo Chang
NIST / ITL
Digital Data Advisor

James St Pierre
NIST / ITL
Deputy Director

NIST Big Data interoperability Framework (NBDIF)

Request for Public Comments (Deadline: September 21, 2017)

The NBD-PWG is actively working to complete version 2 of the set of NBDIF documents. The goals of version 2 are to enhance the version 1 content and define general interfaces between the NIST Big Data Reference Architecture (NBDRA) components by aggregating low-level interactions into high-level general interfaces, and demonstrate how the NBDRA can be used.

Below are the draft **Version 2** documents for your review and comment:

- **Volume 1: Definitions**
  M0635: r1
- **Volume 2: Taxonomies**
  M0636: r1
- **Volume 3: Use Case & Requirements**
  M0637: r1
  M0621 (Use Case Template #2 in PDF): r1, r2
- **Volume 4: Security & Privacy**
  M0638: r1
- **Volume 6: Reference Architecture**
  M0639: r1
- **Volume 7: Standards Roadmap**
  M0640: r1
- **Volume 8: Reference Architecture Interface**
  M0641: r1
- **Volume 9: Adoption and Modernization**
  M0642: r1

Upcoming/Past Events

- 2nd NIST Big Data Workshop, NIST, June 1 & 2, 2017
- IEEE NBD-PWG Workshop, October 27, 2014
- 1st NIST Big Data Workshop, NIST, September 30, 2013

Useful References

- National Privacy Research Strategy, June 24, 2016
- The Federal Big Data Research and Development Strategic Plan, May 19, 2016
- Interim Progress Report on Big Data: Seizing Opportunities, Preserving Values, February, 2015
- PCAST Report on Big Data and Privacy: A Technological Perspective, May, 2014
- Big Data: Seizing Opportunities, Preserving Values, May 1, 2014
- Frontiers in Massive Data Analysis, July, 2013
- Obama Administration Unveils "Big Data" Initiative: Announces $200 Million in New

https://bigdatawg.nist.gov/V2_output_docs.php

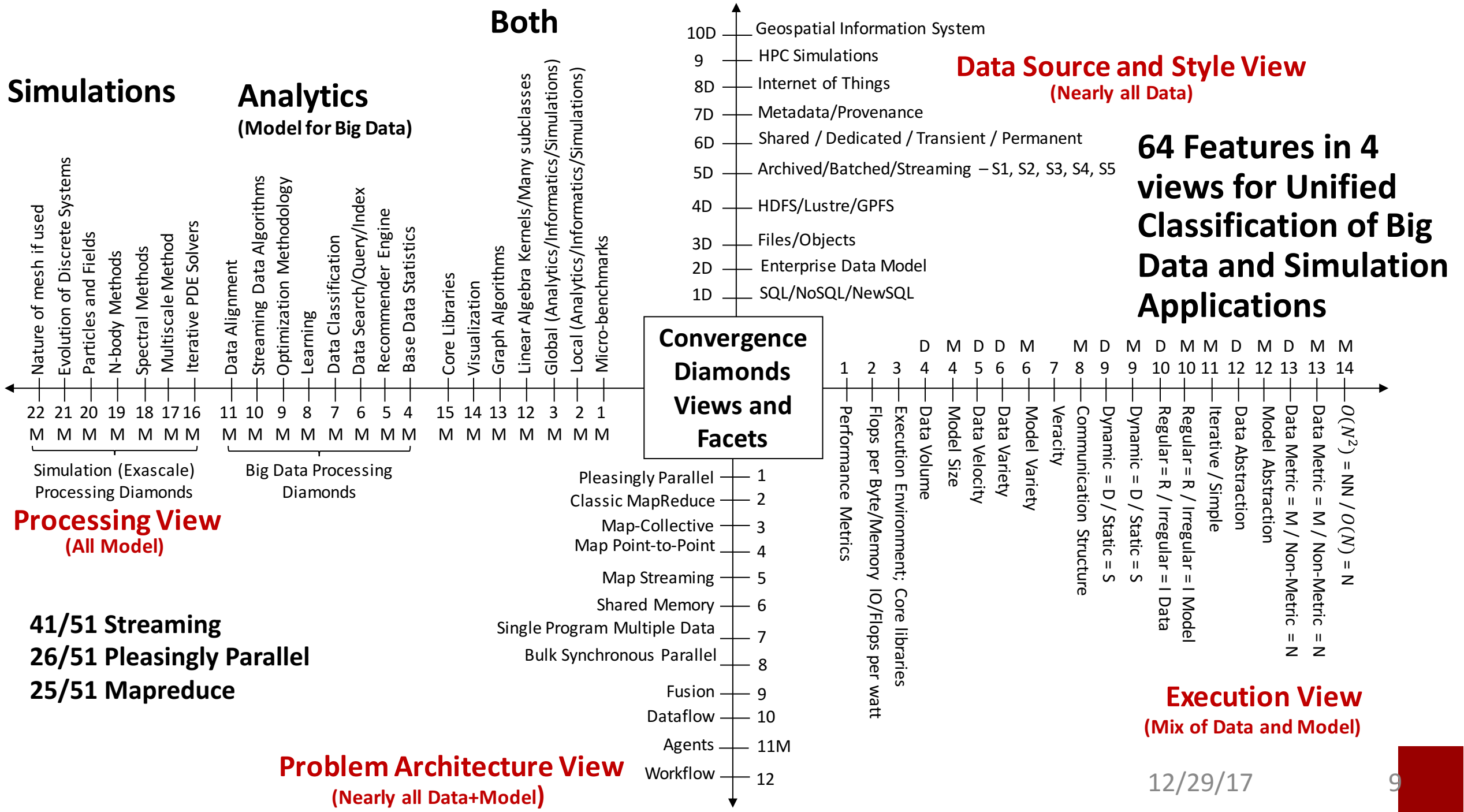
NIST Big Data Public Working Group

Standards

Best Practice

Indiana

Indiana Cloudmesh launching Twister2


Both
Simulations
Analytics
(Model for Big Data)
Nature of mesh if used
Evolution of Discrete Systems
Particles and Fields
N-body Methods
Spectral Methods
Multiscale Method
Iterative PDE Solvers
Data Alignment
Streaming Data Algorithms
Optimization Methodology
Learning
Data Classification
Data Search/Query/Index
Recommender Engine
Base Data Statistics
Core Libraries
Visualization
Graph Algorithms
Linear Algebra Kernels/Many subclasses
Global (Analytics/Informatics/Simulations)
Local (Analytics/Informatics/Simulations)
Micro-benchmarks
22 M 21 M 20 M 19 M 18 M 17 M 16 M
11 M 10 M 9 M 8 M 7 M 6 M 5 M 4 M
15 M 14 M 13 M 12 M 3 M 2 M 1 M
Simulation (Exascale) Processing Diamonds
Big Data Processing Diamonds
Processing View
(All Model)
10D Geospatial Information System
9 HPC Simulations
8D Internet of Things
7D Metadata/Provenance
6D Shared / Dedicated / Transient / Permanent
5D Archived/Batched/Streaming – S1, S2, S3, S4, S5
4D HDFS/Lustre/GPFS
3D Files/Objects
2D Enterprise Data Model
1D SQL/NoSQL/NewSQL
Data Source and Style View
(Nearly all Data)
64 Features in 4 views for Unified Classification of Big Data and Simulation Applications
Convergence Diamonds Views and Facets
1 Performance Metrics
2 Flops per Byte/Memory IO/Flops per watt
3 Execution Environment; Core libraries
D 4 Data Volume
M 4 Model Size
D 5 Data Velocity
D 6 Data Variety
M 6 Model Variety
7 Veracity
M 8 Communication Structure
D 9 Dynamic = D / Static = S
M 9 Dynamic = D / Static = S
D 10 Regular = R / Irregular = I Data
M 10 Regular = R / Irregular = I Model
M 11 Iterative / Simple
D 12 Data Abstraction
M 12 Model Abstraction
D 13 Data Metric = M / Non-Metric = N
M 13 Data Metric = M / Non-Metric = N
M 14 $O(N^2)$ = NN / $O(N)$ = N
Execution View
(Mix of Data and Model)
Pleasingly Parallel 1
Classic MapReduce 2
Map-Collective 3
Map Point-to-Point 4
Map Streaming 5
Shared Memory 6
Single Program Multiple Data 7
Bulk Synchronous Parallel 8
Fusion 9
Dataflow 10
Agents 11M
Workflow 12
Problem Architecture View
(Nearly all Data+Model)
41/51 Streaming
26/51 Pleasingly Parallel
25/51 Mapreduce

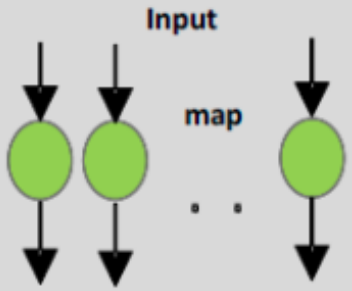
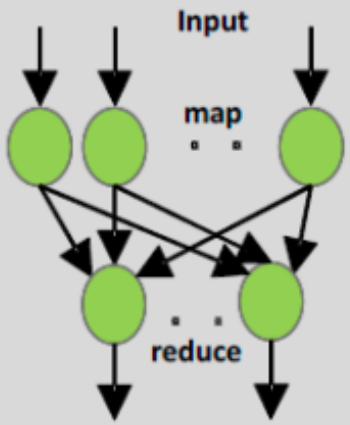
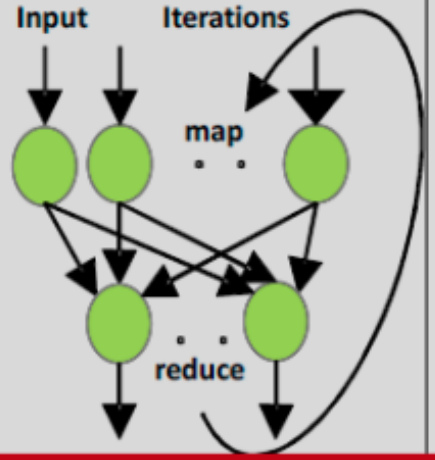
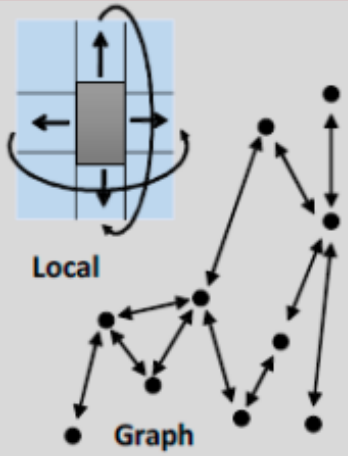
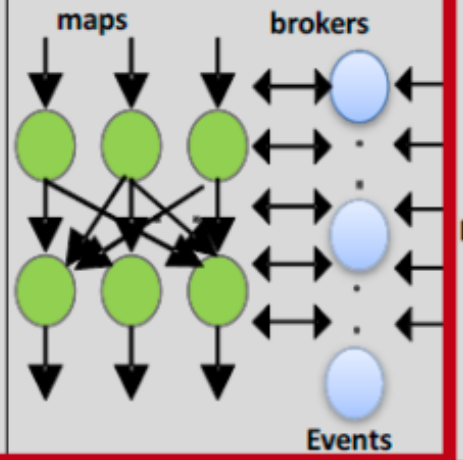
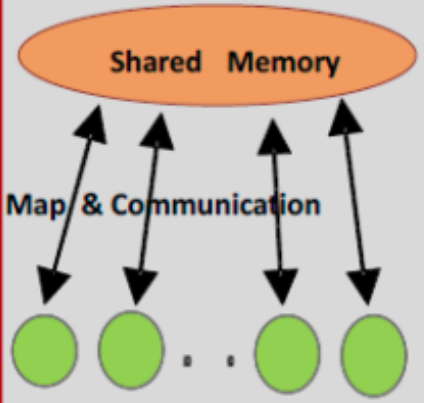
Problem Architecture View (Meta or MacroPatterns)

1. **Pleasingly Parallel** – as in BLAST, Protein docking, some (bio-)imagery including **Local Analytics or Machine Learning** – ML or filtering pleasingly parallel, as in bio-imagery, radar images (pleasingly parallel but sophisticated local analytics)
2. **Classic MapReduce:** Search, Index and Query and Classification algorithms like collaborative filtering (G1 for MRStat in Features, G7)
3. **Map-Collective:** Iterative maps + communication dominated by "collective" operations as in reduction, broadcast, gather, scatter. Common datamining pattern
4. **Map-Point to Point:** Iterative maps + communication dominated by many small point to point messages as in graph algorithms
5. **Map-Streaming:** Describes streaming, steering and assimilation problems
6. **Shared Memory:** Some problems are asynchronous and are easier to parallelize on shared rather than distributed memory – see some graph algorithms
7. **SPMD:** Single Program Multiple Data, common parallel programming feature
8. **BSP or Bulk Synchronous Processing:** well-defined compute-communication phases
9. **Fusion:** Knowledge discovery often involves fusion of multiple methods.
10. **Dataflow:** Important application features often occurring in composite Ogres
11. **Use Agents:** as in epidemiology (swarm approaches) This is **Model** only
12. **Workflow:** All applications often involve orchestration (workflow) of multiple components

Most (11 of total 12) are properties of Data+Model

Six Computation Paradigms for Data Analytics

Note Problem and System Architecture as efficient execution says they must match

(1) Map Only Pleasingly Parallel	(2) Classic Map-Reduce	(3) Iterative Map Reduce or Map-Collective	(4) Point to Point or Map-Communication	(5) Map-Streaming	(6) Shared memory Map-Communication
					
- BLAST Analysis - Local Machine Learning - Pleasingly Parallel	- High Energy Physics (HEP) Histograms, - Web search - Recommender Engines	- Expectation Maximization - Clustering - Linear Algebra - PageRank Global Machine	- Classic MPI - PDE Solvers and Particle Dynamics - Graph Learning	- Streaming images from Synchrotron sources, Telescopes, Internet of Things	- Difficult to parallelize - asynchronous parallel Graph

Classic Cloud Workload

These 3 are focus of Twister2 but we need to preserve capability on first 2 paradigms

Data and Model in Big Data and Simulations I

- Need to discuss **Data** and **Model** as problems have both intermingled, but we can get insight by separating which allows better understanding of **Big Data - Big Simulation "convergence" (or differences!)**
- The **Model** is a user construction and it has a "**concept**", **parameters** and gives **results** determined by the computation. We use term "model" in a general fashion to cover all of these.
- **Big Data** problems can be broken up into **Data** and **Model**
  - For **clustering,** the model parameters are cluster centers while the data is set of points to be clustered
  - For **queries**, the model is structure of database and results of this query while the data is whole database queried and SQL query
  - For **deep learning** with ImageNet, the model is chosen network with model parameters as the network link weights. The data is set of images used for training or classification

Data and Model in Big Data and Simulations II

- **Simulations** can also be considered as **Data** plus **Model**
  - **Model** can be formulation with particle dynamics or partial differential equations defined by parameters such as particle positions and discretized velocity, pressure, density values
  - **Data** could be small when just boundary conditions
  - **Data** large with data assimilation (weather forecasting) or when data visualizations are produced by simulation
- **Big Data** implies Data is large but Model varies in size
  - e.g. **LDA** (Latent Dirichlet Allocation) with many topics or **deep learning** has a large model
  - **Clustering** or **Dimension reduction** can be quite small in model size
- **Data** often static between iterations (unless streaming); **Model parameters** vary between iterations
- **Data** and **Model Parameters** are often **confused** in papers as term data used to describe the parameters of models.
- **Models** in **Big Data** and **Simulations** have many similarities and allow **convergence**

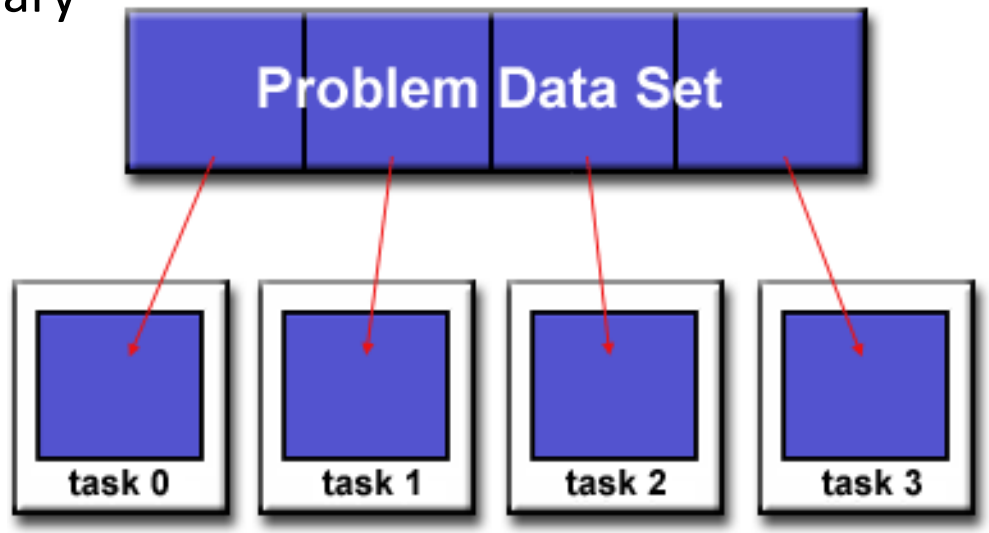
Convergence/Divergence Points for HPC-Cloud-Edge- Big Data-Simulation

- **Applications** – Divide use cases into **Data** and **Model** and compare characteristics separately in these two components with 64 Convergence Diamonds (features).
  - Identify importance of streaming data, pleasingly parallel, global/local machine-learning
- **Software** – Single model of **High Performance Computing (HPC) Enhanced Big Data Stack HPC-ABDS**. 21 Layers adding high performance runtime to Apache systems **HPC-FaaS** Programming Model
  - **Serverless** Infrastructure as a Service IaaS
- **Hardware** system designed for functionality and performance of application type e.g. disks, interconnect, memory, CPU acceleration different for machine learning, pleasingly parallel, data management, streaming, simulations
  - Use DevOps to automate deployment of event-driven software defined systems on hardware: **HPCCloud 2.0**
- **Total System Solutions (wisdom) as a Service: HPCCloud 3.0**

Uses DevOps not discussed in this talk

Parallel Computing: Big Data and Simulations

- **All the different programming models** (Spark, Flink, Storm, Naiad, MPI/OpenMP) have the **same high level approach** but application requirements and system architecture can give different appearance
- First: Break Problem **Data** and/or **Model-parameters** into parts assigned to separate nodes, processes, threads
- Then: In parallel, do computations typically leaving data untouched but changing model-parameters. Called **Maps** in MapReduce parlance; typically **owner computes rule**.
- If **Pleasingly parallel**, that's all it is except for management
- If **Globally parallel,** need to communicate results of computations between nodes during job
- Communication mechanism (TCP, RDMA, Native Infiniband) can vary
- Communication Style (Point to Point, Collective, Pub-Sub) can vary
- Possible need for sophisticated dynamic changes in partioning (load balancing)
- Computation either on fixed tasks or flow between tasks
- Choices: "Automatic Parallelism or Not"
- Choices: "Complicated Parallel Algorithm or Not"
- Fault-Tolerance model can vary
- Output model can vary: RDD or Files or Pipes



Difficulty in Parallelism

Size of Synchronization constraints

Disk I/O

Loosely Coupled

MapReduce as in scalable databases

Pleasingly Parallel
Often independent events

HPC Clouds
High Performance Interconnect

Global Machine Learning
e.g. parallel clustering

Deep Learning

HPC Clouds/Supercomputers
Memory access also critical

Unstructured Adaptive Sparsity
Medium size Jobs

LDA

Graph Analytics e.g. subgraph mining

Linear Algebra at core
(typically not sparse)

Large scale simulations

Current major Big Data category

Structured Adaptive Sparsity
Huge Jobs

Commodity Clouds

Exascale Supercomputers

Spectrum of Applications and Algorithms

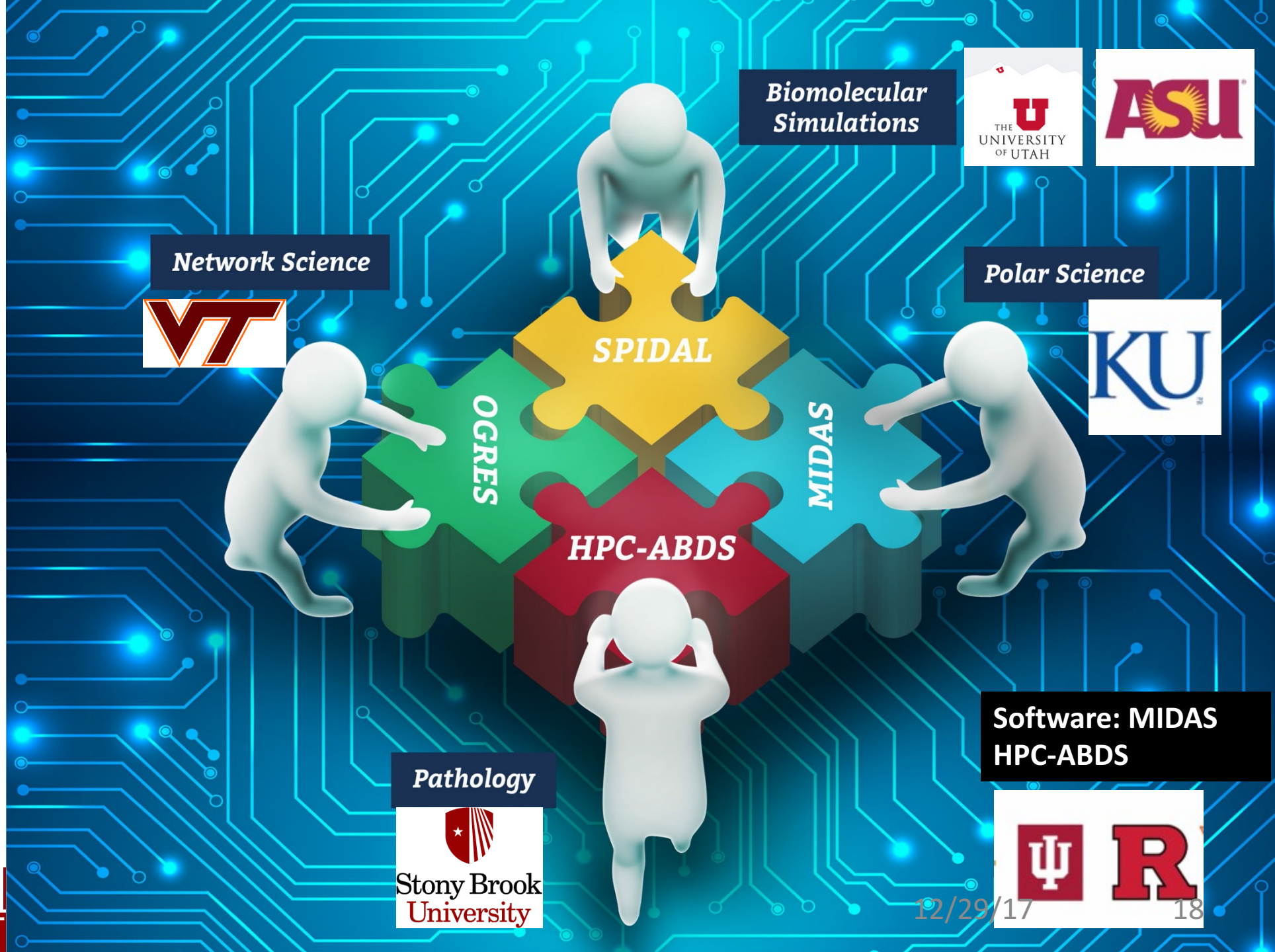
Software
HPC-ABDS
HPC-FaaS

NSF 1443054: CIF21 DIBBs: Middleware and High Performance Analytics Libraries for Scalable Data Science

Ogres Application Analysis

HPC-ABDS and HPC-FaaS Software Harp and Twister2 Building Blocks

SPIDAL Data Analytics Library



HPC-ABDS

Integrated wide range of HPC and Big Data technologies.

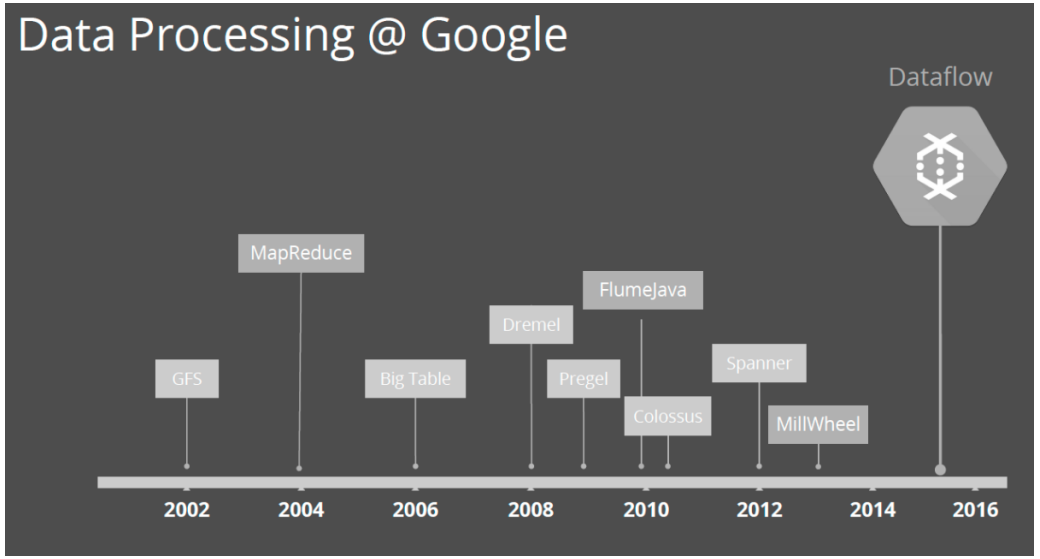
I gave up updating list in January 2016!

Kaleidoscope of (Apache) Big Data Stack (ABDS) and HPC Technologies

Cross-Cutting Functions	
1) **Message and Data Protocols:** Avro, Thrift, Protobuf	17) **Workflow-Orchestration:** ODE, ActiveBPEL, Airavata, Pegasus, Kepler, Swift, Taverna, Triana, Trident, BioKepler, Galaxy, IPython, Dryad, Naiad, Oozie, Tez, Google FlumeJava, Crunch, Cascading, Scalding, e-Science Central, Azure Data Factory, Google Cloud Dataflow, NiFi (NSA), Jitterbit, Talend, Pentaho, Apatar, Docker Compose, KeystoneML
2) **Distributed Coordination :** Google Chubby, Zookeeper, Giraffe, JGroups	16) **Application and Analytics:** Mahout , MLlib , MLbase, DataFu, R, pbdR, Bioconductor, ImageJ, OpenCV, Scalapack, PetSc, PLASMA MAGMA, Azure Machine Learning, Google Prediction API & Translation API, mlpy, scikit-learn, PyBrain, CompLearn, DAAL(Intel), Caffe, Torch, Theano, DL4j, H2O, IBM Watson, Oracle PGX, GraphLab, GraphX, IBM System G, GraphBuilder(Intel), TinkerPop, Parasol, Dream:Lab, Google Fusion Tables, CINET, NWB, Elasticsearch, Kibana, Logstash, Graylog, Splunk, Tableau, D3.js, three.js, Potree, DC.js, TensorFlow, CNTK
3) **Security & Privacy:** InCommon, Eduroam OpenStack Keystone, LDAP, Sentry, Sqrrl, OpenID, SAML OAuth	15B) **Application Hosting Frameworks:** Google App Engine, AppScale, Red Hat OpenShift, Heroku, Aerobatic, AWS Elastic Beanstalk, Azure, Cloud Foundry, Pivotal, IBM BlueMix, Ninefold, Jelastic, Stackato, appfog, CloudBees, Engine Yard, CloudControl, dotCloud, Dokku, OSGi, HUBzero, OODT, Agave, Atmosphere 15A) **High level Programming:** Kite, Hive, HCatalog, Tajo, Shark, Phoenix, Impala, MRQL, SAP HANA, HadoopDB, PolyBase, Pivotal HD/Hawq, Presto, Google Dremel, Google BigQuery, Amazon Redshift, Drill, Kyoto Cabinet, Pig, Sawzall, Google Cloud DataFlow, Summingbird
4) **Monitoring:** Ambari, Ganglia, Nagios, Inca	14B) **Streams:** Storm, S4, Samza, Granules, Neptune, Google MillWheel, Amazon Kinesis, LinkedIn, Twitter Heron, Databus, Facebook Puma/Ptail/Scribe/ODS, Azure Stream Analytics, Floe, Spark Streaming, Flink Streaming, DataTurbine 14A) **Basic Programming model and runtime**, **SPMD, MapReduce:** Hadoop, Spark, Twister, MR-MPI, Stratosphere (Apache Flink), Reef, Disco, Hama, Giraph, Pregel, Pegasus, Ligra, GraphChi, Galois, Medusa-GPU, MapGraph, Totem
21 layers Over 350 Software Packages	13) **Inter process communication Collectives, point-to-point, publish-subscribe:** MPI, HPX-5, Argo BEAST HPX-5 BEAST PULSAR, Harp, Netty, ZeroMQ, ActiveMQ, RabbitMQ, NaradaBrokering, QPid, Kafka, Kestrel, JMS, AMQP, Stomp, MQTT, Marionette Collective, **Public Cloud:** Amazon SNS, Lambda, Google Pub Sub, Azure Queues, Event Hubs
January 29 2016	12) **In-memory databases/caches:** Gora (general object from NoSQL), Memcached, Redis, LMDB (key value), Hazelcast, Ehcache, Infinispan, VoltDB, H-Store
	12) **Object-relational mapping:** Hibernate, OpenJPA, EclipseLink, DataNucleus, ODBC/JDBC
	12) **Extraction Tools:** UIMA, Tika
	11C) **SQL(NewSQL):** Oracle, DB2, SQL Server, SQLite, MySQL, PostgreSQL, CUBRID, Galera Cluster, SciDB, Rasdaman, Apache Derby, Pivotal Greenplum, Google Cloud SQL, Azure SQL, Amazon RDS, Google F1, IBM dashDB, N1QL, BlinkDB, Spark SQL
	11B) **NoSQL:** Lucene, Solr, Solandra, Voldemort, Riak, ZHT, Berkeley DB, Kyoto/Tokyo Cabinet, Tycoon, Tyrant, MongoDB, Espresso, CouchDB, Couchbase, IBM Cloudant, Pivotal Gemfire, HBase, Google Bigtable, LevelDB, Megastore and Spanner, Accumulo, Cassandra, RYA, Sqrrl, Neo4J, graphdb, Yarcdata, AllegroGraph, Blazegraph, Facebook Tao, Titan:db, Jena, Sesame **Public Cloud:** Azure Table, Amazon Dynamo, Google DataStore
	11A) **File management:** iRODS, NetCDF, CDF, HDF, OPeNDAP, FITS, RCFile, ORC, Parquet
	10) **Data Transport:** BitTorrent, HTTP, FTP, SSH, Globus Online (GridFTP), Flume, Sqoop, Pivotal GPLOAD/GPFDIST
	9) **Cluster Resource Management**: Mesos, Yarn, Helix, Llama, Google Omega, Facebook Corona, Celery, HTCondor, SGE, OpenPBS, Moab, Slurm, Torque, Globus Tools, Pilot Jobs
	8) **File systems:** HDFS, Swift, Haystack, f4, Cinder, Ceph, FUSE, Gluster, Lustre, GPFS, GFFS **Public Cloud:** Amazon S3, Azure Blob, Google Cloud Storage
	7) **Interoperability:** Libvirt, Libcloud, JClouds, TOSCA, OCCI, CDMI, Whirr, Saga, Genesis
	6) **DevOps:** Docker (Machine, Swarm), Puppet, Chef, Ansible, SaltStack, Boto, Cobbler, Xcat, Razor, CloudMesh, Juju, Foreman, OpenStack Heat, Sahara, Rocks, Cisco Intelligent Automation for Cloud, Ubuntu MaaS, Facebook Tupperware, AWS OpsWorks, OpenStack Ironic, Google Kubernetes, Buildstep, Gitreceive, OpenTOSCA, Winery, CloudML, Blueprints, Terraform, DevOpSlang, Any2Api
	5) **IaaS Management from HPC to hypervisors:** Xen, KVM, QEMU, Hyper-V, VirtualBox, OpenVZ, LXC, Linux-Vserver, OpenStack, OpenNebula, Eucalyptus, Nimbus, CloudStack, CoreOS, rkt, VMware ESXi, vSphere and vCloud, Amazon, Azure, Google and other public Clouds **Networking:** Google Cloud DNS, Amazon Route 53

Components of Big Data Stack

- Google likes to show a timeline; we can build on (Apache version of) this
- 2002 **Google File System** GFS ~HDFS (Level 8)
- 2004 **MapReduce** Apache Hadoop (Level 14A)
- 2006 **Big Table** Apache Hbase (Level 11B)
- 2008 **Dremel** Apache Drill (Level 15A)
- 2009 **Pregel** Apache Giraph (Level 14A)
- 2010 **FlumeJava** Apache Crunch (Level 17)
- 2010 **Colossus** better GFS (Level 18)

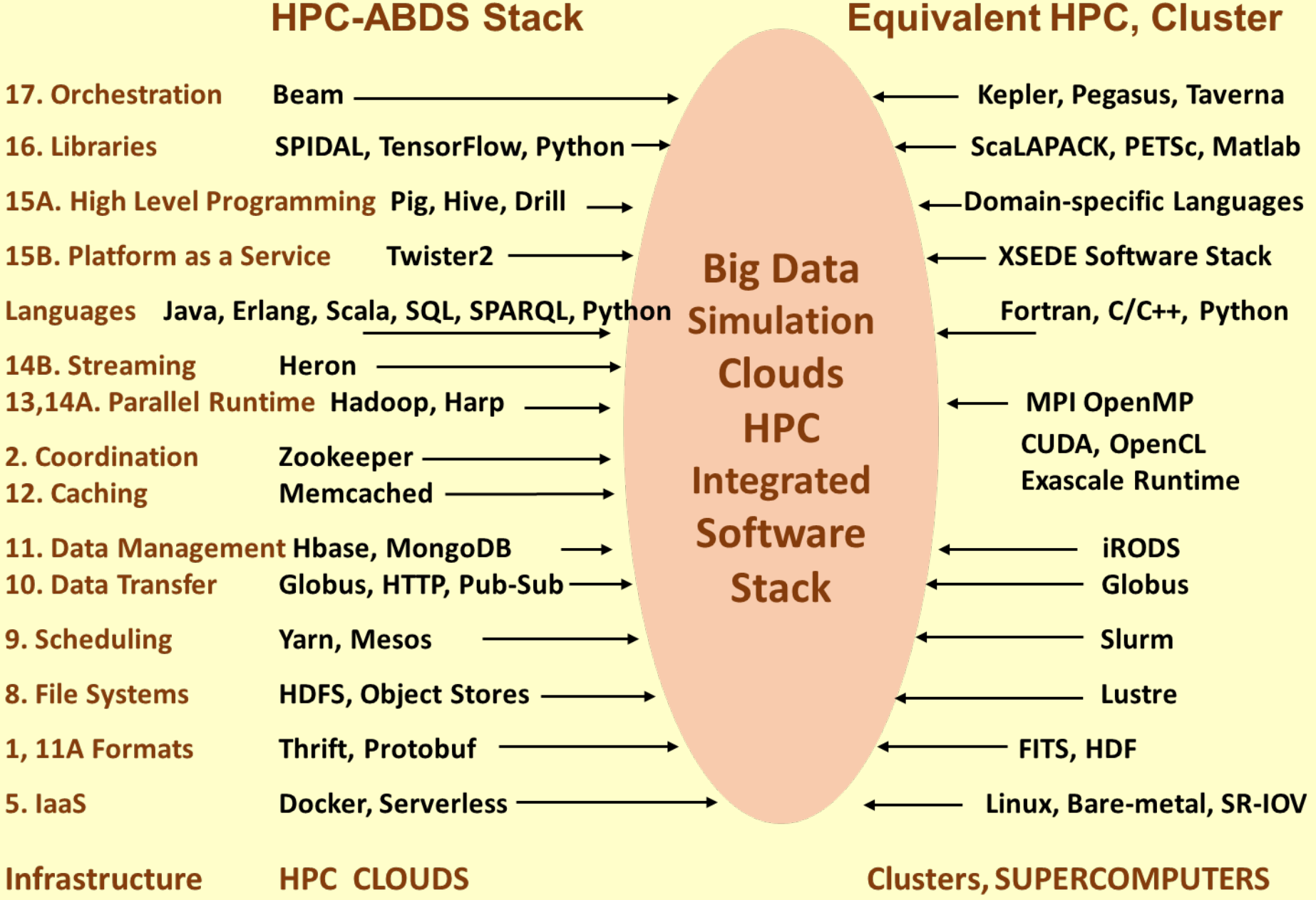


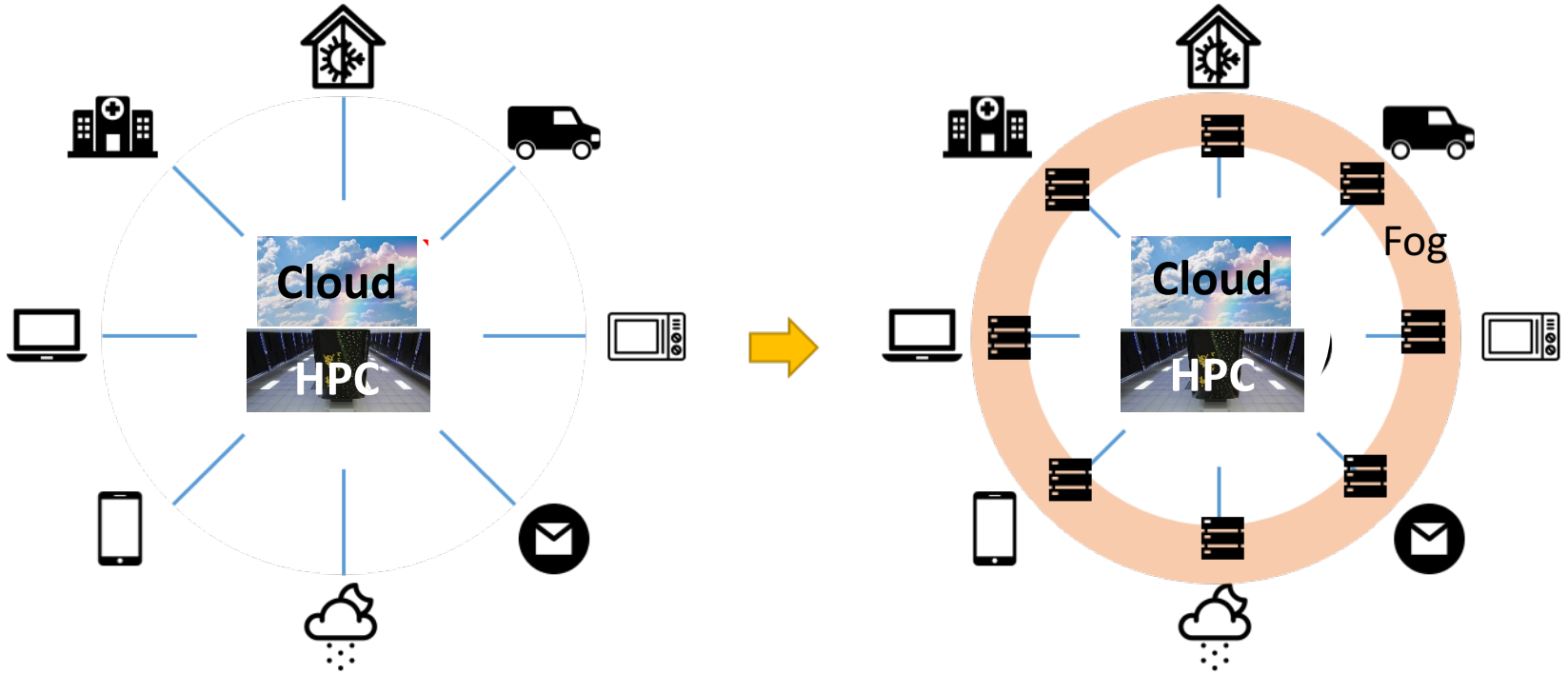
- 2012 **Spanner** horizontally scalable NewSQL database ~CockroachDB (Level 11C)
- 2013 **F1** horizontally scalable SQL database (Level 11C)
- 2013 **MillWheel** ~Apache Storm, Twitter Heron (Google not first!) (Level 14B)
- 2015 **Cloud Dataflow** Apache Beam with Spark or Flink (dataflow) engine (Level 17)
- Functionalities not identified: **Security(3), Data Transfer(10), Scheduling(9), DevOps(6), serverless computing** (where Apache has **OpenWhisk) (5)**

HPC-ABDS Levels in ()

Different choices in software systems in Clouds and HPC. HPC-ABDS takes cloud software augmented by HPC when needed to improve performance

16 of 21 layers plus languages





Centralized HPC Cloud + IoT Devices

Centralized HPC Cloud + Edge = Fog + IoT Devices

HPC Cloud can be federated

Implementing Twister2 to support a Grid linked to an HPC Cloud

Twister2: "Next Generation Grid - Edge – HPC Cloud"

- Original 2010 **Twister** paper has 914 citations; it was a particular approach to MapCollective iterative processing for machine learning
- **Re-engineer** current Apache Big Data and HPC software systems as **a toolkit**
- Support a **serverless (cloud-native) dataflow event-driven HPC-FaaS (microservice)** framework running across application and geographic domains.
  - Support all types of Data analysis from GML to Edge computing
- Build on Cloud best practice but use HPC wherever possible to get high performance
- Smoothly support current paradigms Hadoop, Spark, Flink, Heron, MPI, DARMA …
- Use **interoperable** common abstractions but multiple **polymorphic** implementations.
  - i.e. do not require a single runtime
- Focus on Runtime but this implies HPC-FaaS programming and execution model
- This defines **a next generation Grid** based on data and edge devices – not computing as in old Grid

See paper http://dsc.soic.indiana.edu/publications/twister2_design_big_data_toolkit.pdf

Proposed Twister2 Approach

- Unit of Processing is an **Event driven Function** (a **microservice**) replaces libraries
  - Can have state that may need to be preserved in place (Iterative MapReduce)
  - Functions can be single or 1 of 100,000 maps in large parallel code
- Processing units run in **HPC clouds, fogs** or **devices** but these all have similar software architecture (see AWS Greengrass and Lambda)
  - Universal Programming model so **Fog (e.g. car) looks like a cloud to a device (radar sensor) while public cloud looks like a cloud to the fog (car)**
- Analyze the **runtime of existing systems (More study needed)**
  - Hadoop, Spark, Flink, Pregel Big Data Processing
  - Storm, Heron Streaming Dataflow
  - Kepler, Pegasus, NiFi workflow systems
  - Harp Map-Collective, MPI and HPC AMT runtime like DARMA
  - And approaches such as GridFTP and CORBA/HLA (!) for wide area data links

Comparing Spark Flink Heron and MPI

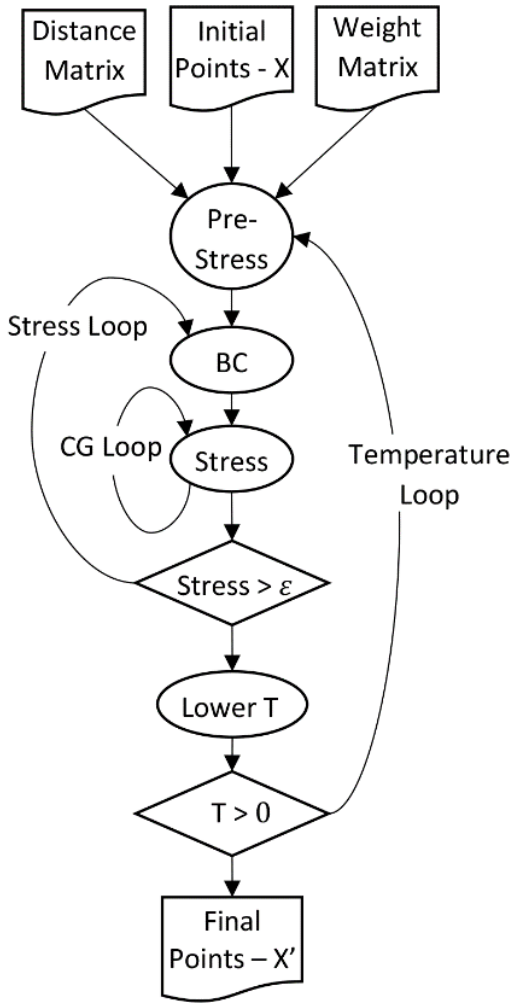
- On Global Machine Learning GML.
- Note I said Spark and Flink are successful on LML not GML and currently LML is more common than GML

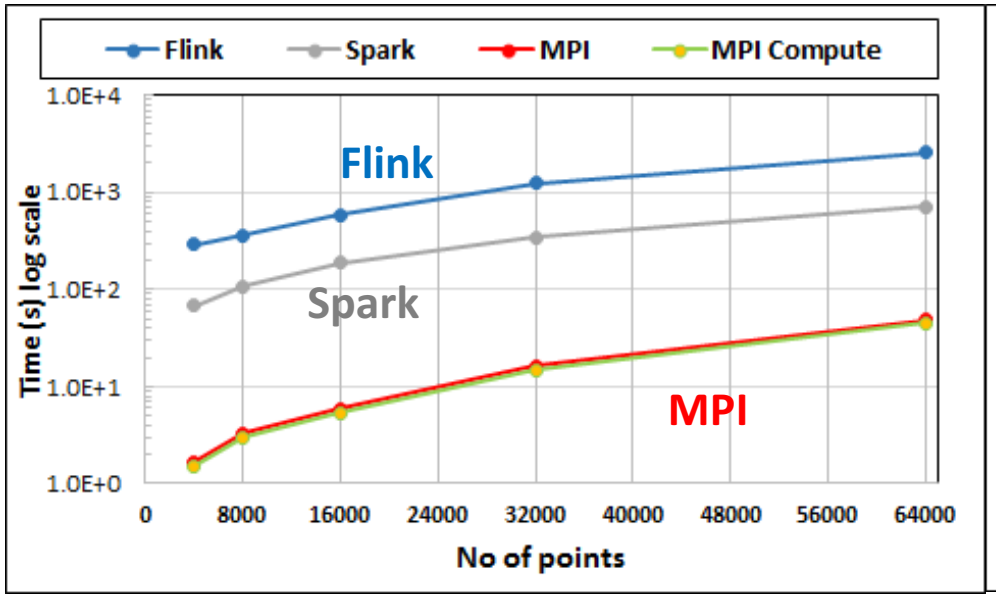
Machine Learning with MPI, Spark and Flink

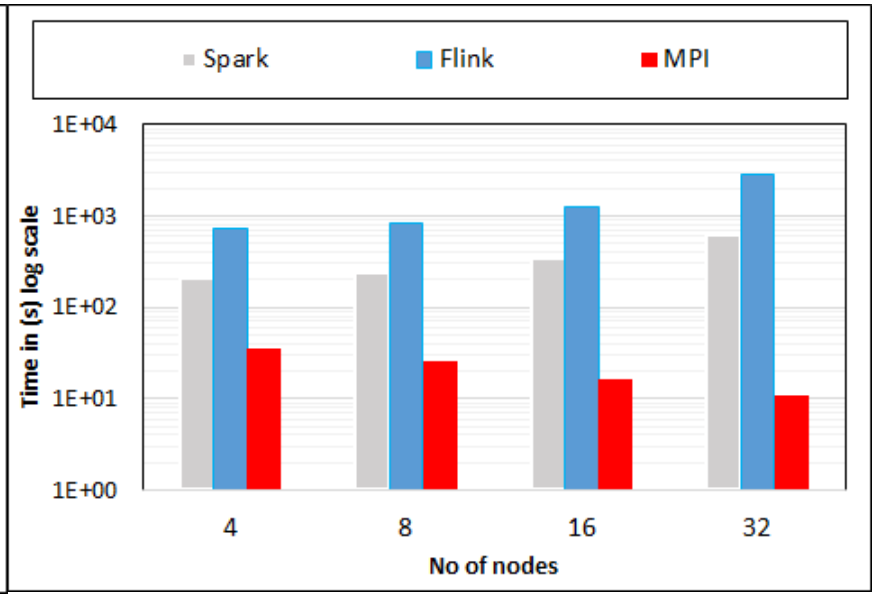
- Three algorithms implemented in three runtimes
  - Multidimensional Scaling (MDS)
  - Terasort
  - K-Means (drop as no time)
- Implementation in Java
  - MDS is the most complex algorithm - three nested parallel loops
  - K-Means - one parallel loop
  - Terasort - no iterations

- With care, Java performance ~ C performance
- Without care, Java performance << C performance (details omitted)

Multidimensional Scaling: 3 Nested Parallel Sections







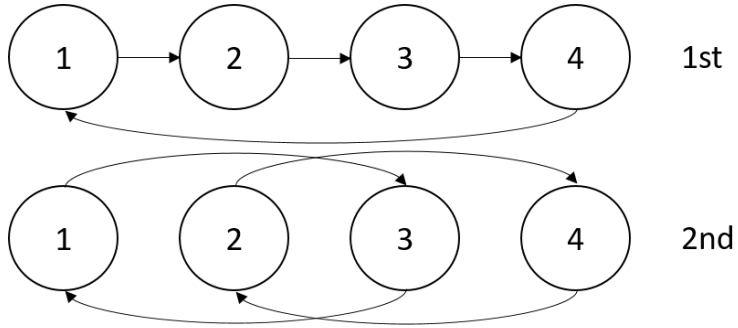
MPI Factor of 20-200 Faster than Spark/Flink

MDS execution time on **16 nodes** with 20 processes in each node with varying number of points

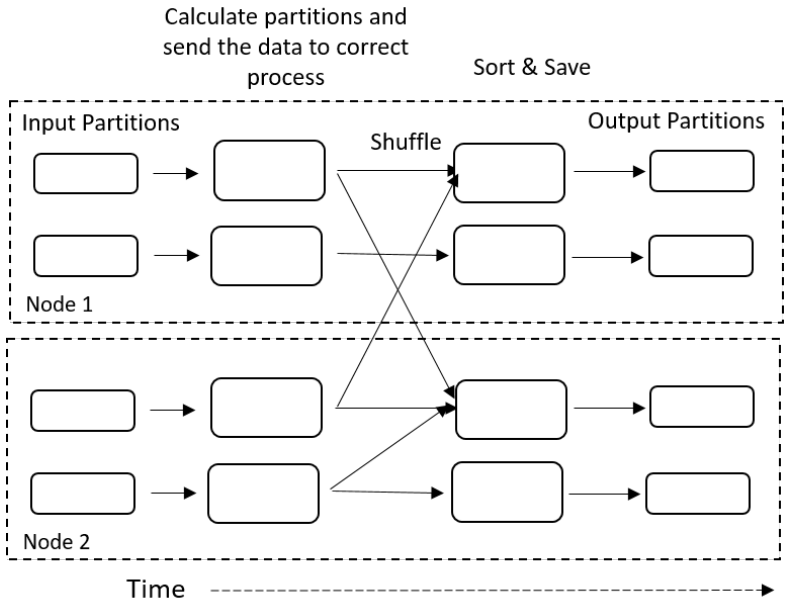
MDS execution time with 32000 points on **varying number of nodes**. Each node runs 20 parallel tasks

Terasort

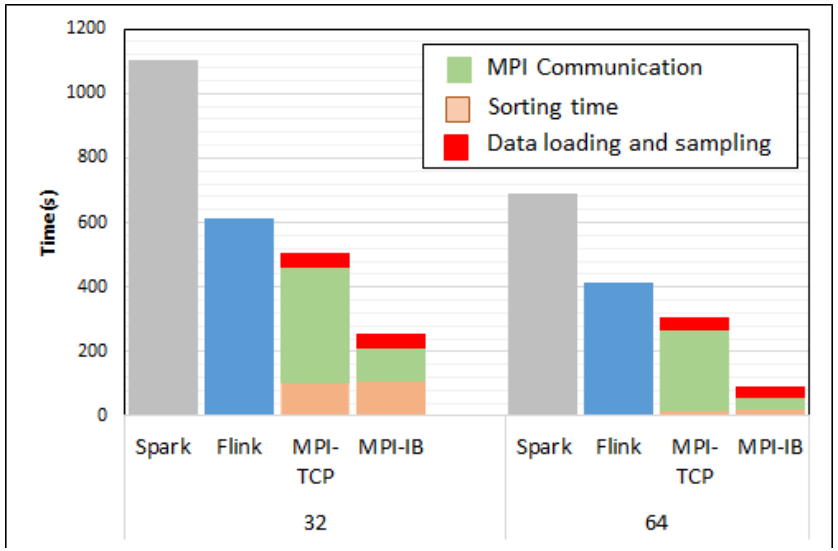
Transfer data using MPI



Sorting 1TB of data records

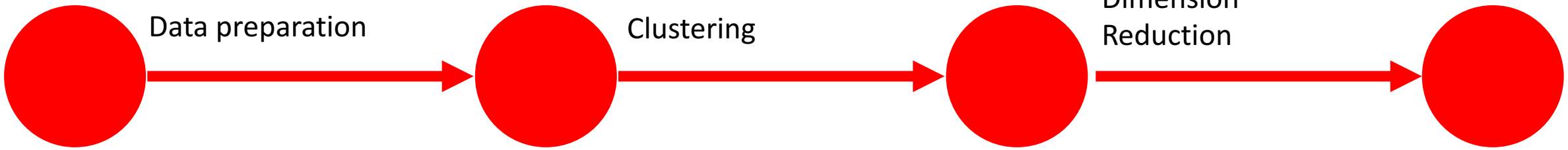


Partition the data using a sample and regroup

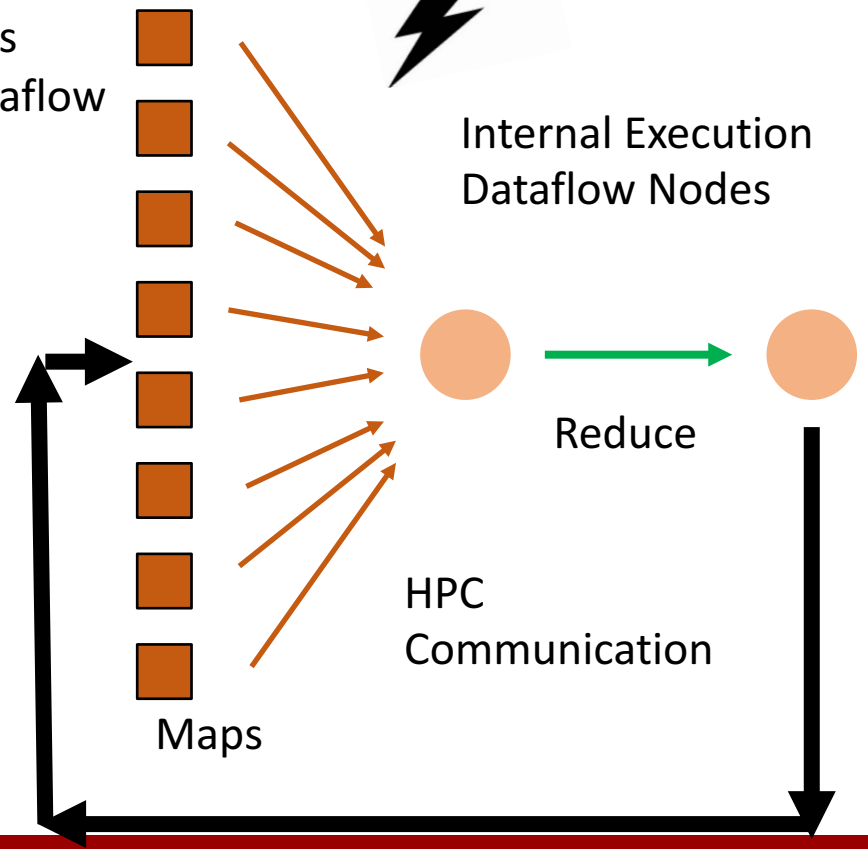


Terasort execution time in 64 and 32 nodes. Only MPI shows the sorting time and communication time as other two frameworks doesn't provide a viable method to accurately measure them. Sorting time includes data save time. MPI-IB - MPI with Infiniband

Coarse Grain Dataflows links jobs in such a pipeline



But internally to each job you can also elegantly express algorithm as dataflow but with more stringent performance constraints



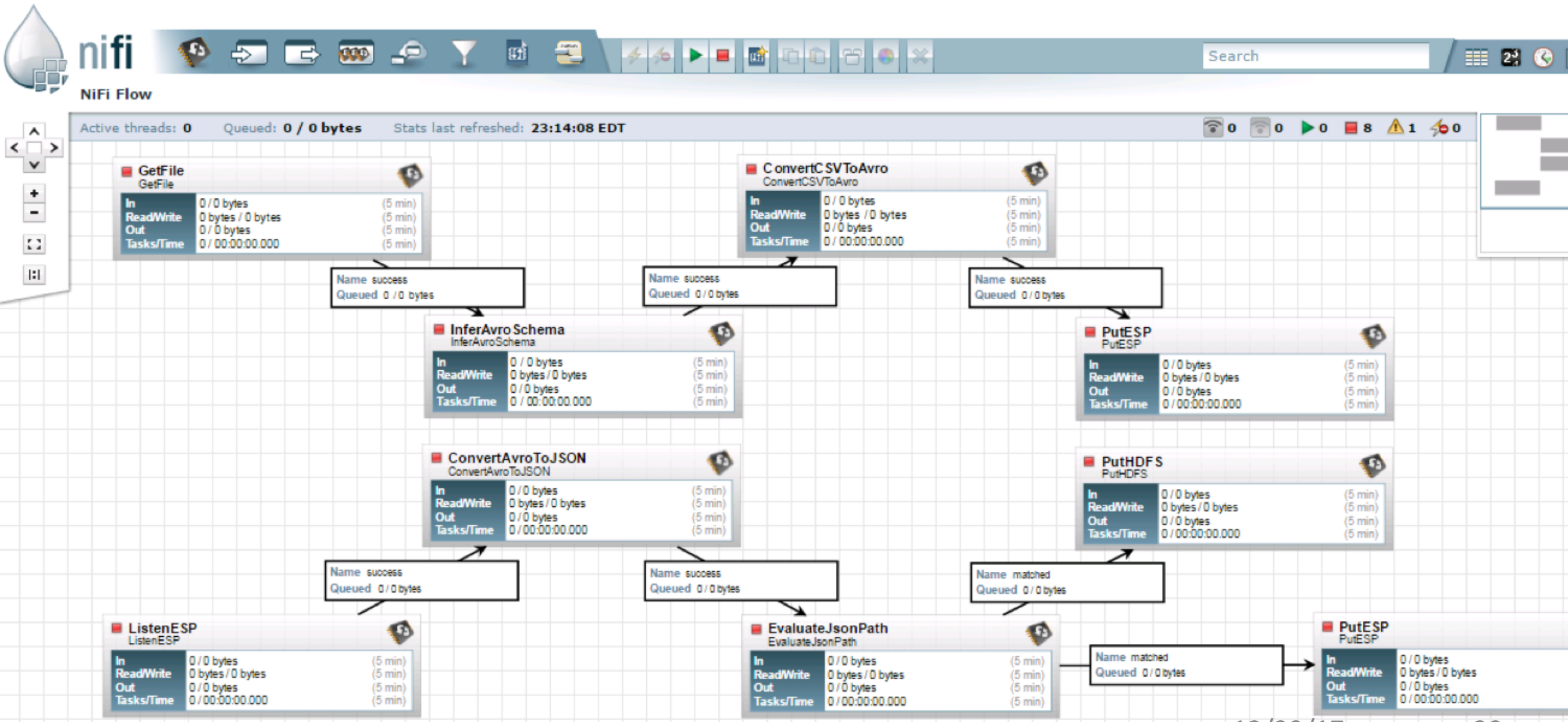
Corresponding to classic Spark K-means Dataflow

- P = loadPoints()
- C = loadInitCenters()
- for (int i = 0; i < 10; i++) {
- T = P.map().withBroadcast(C)
- C = T.reduce() }

Iterate

Dataflow at Different Grain sizes

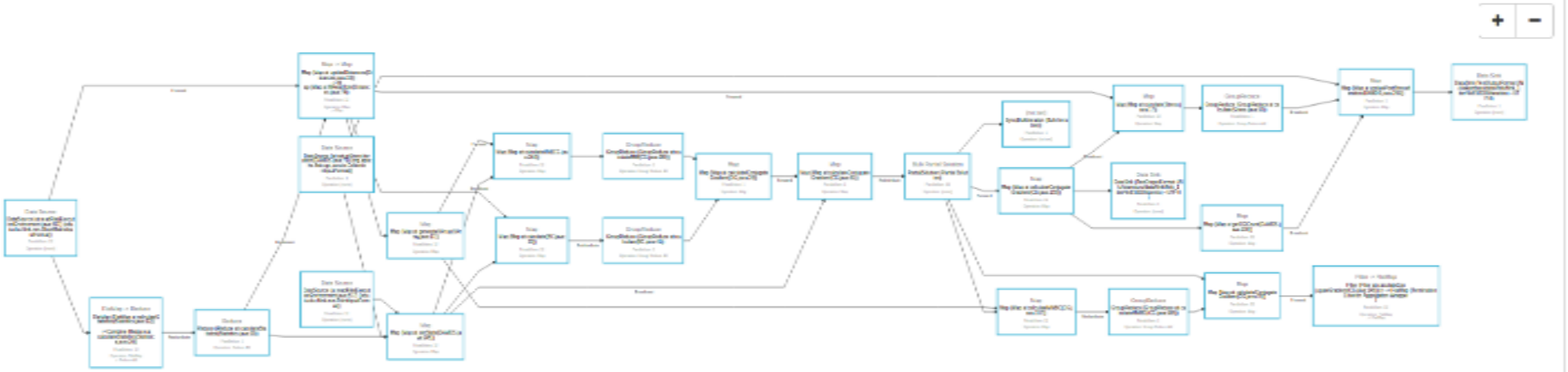
NiFi Workflow



Flink MDS Dataflow Graph

Flink Java Job at Mon Oct 03 12:27:10 EDT 2016 | b3057e083f05a3b5c4ee2f515ab7d7b8 | 2016-10-03, 12:27:10 - 2016-10-03, 12:39:34 | 12m 23s

Plan | Timeline | Exceptions | Properties | Configuration



Subtasks | TaskManagers | Accumulators | Checkpoints

Start Time	End Time	Duration	Name	Bytes received	Records received	Bytes sent	Records sent	Tasks	Status
2016-10-03, 12:27:10	2016-10-03, 12:33:18	6m 7s	DataSource (at readFile(ExecutionEnvironment.java:517) (edu.iu.dsc.flink.mm.ShortMatrixInputFormat))	0 B	0	3.81 GB	64	0 0 0 32 0 0 0	FINISHED
2016-10-03, 12:27:10	2016-10-03, 12:27:10	45ms	DataSource (at setupStressIteration(DAMDS.java:71) (org.apache.flink.api.java.io.CollectionInputFormat))	0 B	0	1.68 KB	33	0 0 0 1 0 0 0	FINISHED
2016-10-03, 12:27:10	2016-10-03, 12:27:11	739ms	DataSource (at readFile(ExecutionEnvironment.java:517) (edu.iu.dsc.flink.mm.PointInputFormat))	0 B	0	750 KB	1	0 0 0 32 0 0 0	FINISHED
2016-10-03, 12:33:18	2016-10-03, 12:33:23	5s	CHAIN FlatMap (FlatMap at calculateStatistics(Statistics.java:12)) -> Combine (Reduce at calculateStatistics(Statistics.java:20))	1.91 GB	32	2.56 KB	32	0 0 0 32 0 0 0	FINISHED
2016-10-03, 12:33:23	2016-10-03, 12:33:23	426ms	Reduce (Reduce at calculateStatistics(Statistics.java:20))	2.56 KB	32	5.13 KB	64	0 0 0 1 0 0 0	FINISHED
2016-10-03, 12:33:18	2016-10-03, 12:33:57	38s	CHAIN Map (Map at updateDistances(Distances.java:33)) -> Map (Map at filReadJoin(Distances.java:74))	1.91 GB	32	11.4 GB	96	0 0 0 32 0 0 0	FINISHED
2016-10-03, 12:33:57	2016-10-03, 12:34:29	32s	Map (Map at generateVArray(VArray.java:17))	3.81 GB	32	3.82 GB	64	0 0 0 32 0 0 0	FINISHED
2016-10-03, 12:27:10	2016-10-03, 12:33:24	6m 13s	Map (Map at joinStats(DAMDS.java:345))	750 KB	1	47.6 MB	65	0 0 0 32 0 0 0	FINISHED
2016-10-03, 12:33:24	2016-10-03, 12:34:40	1m 16s	Map (Map at calculateMM(CG.java:260))	1.91 GB	32	752 KB	32	0 0 0 32 0 0 0	FINISHED

http://www.iterativemapreduce.org/

Implementing Twister2 in detail I

This breaks rule from 2012-2017 of not “competing” with but rather “enhancing” Apache

Look at Communication in detail

Twister2 Components I

Area	Component	Implementation	Comments: User API
Architecture Specification	Coordination Points	State and Configuration Management; Program, Data and Message Level	Change execution mode; save and reset state
	Execution Semantics	Mapping of Resources to Bolts/Maps in Containers, Processes, Threads	Different systems make different choices - why?
	Parallel Computing	Spark Flink Hadoop Pregel MPI modes	Owner Computes Rule
Job Submission	(Dynamic/Static) Resource Allocation	Plugins for Slurm, Yarn, Mesos, Marathon, Aurora	Client API (e.g. Python) for Job Management
Task System	Task migration	Monitoring of tasks and migrating tasks for better resource utilization	Task-based programming with Dynamic or Static Graph API; FaaS API; Support accelerators (CUDA,KNL)
	Elasticity	OpenWhisk	
	Streaming and FaaS Events	Heron, OpenWhisk, Kafka/RabbitMQ	
	Task Execution	Process, Threads, Queues	
	Task Scheduling	Dynamic Scheduling, Static Scheduling, Pluggable Scheduling Algorithms	
	Task Graph	Static Graph, Dynamic Graph Generation	

Twister2 Components II

Area	Component	Implementation	Comments
Communication API	Messages	Heron	This is user level and could map to multiple communication systems
	Dataflow Communication	Fine-Grain Twister2 Dataflow communications: MPI,TCP and RMA Coarse grain Dataflow from NiFi, Kepler?	Streaming, ETL data pipelines; Define new Dataflow communication API and library
	BSP Communication Map-Collective	Conventional MPI, Harp	MPI Point to Point and Collective API
Data Access	Static (Batch) Data	File Systems, NoSQL, SQL	Data API
	Streaming Data	Message Brokers, Spouts	
Data Management	Distributed Data Set	Relaxed Distributed Shared Memory(immutable data), Mutable Distributed Data	Data Transformation API; Spark RDD, Heron Streamlet
Fault Tolerance	Check Pointing	Upstream (streaming) backup; Lightweight; Coordination Points; Spark/Flink, MPI and Heron models	Streaming and batch cases distinct; Crosses all components
Security	Storage, Messaging, execution	Research needed	Crosses all Components

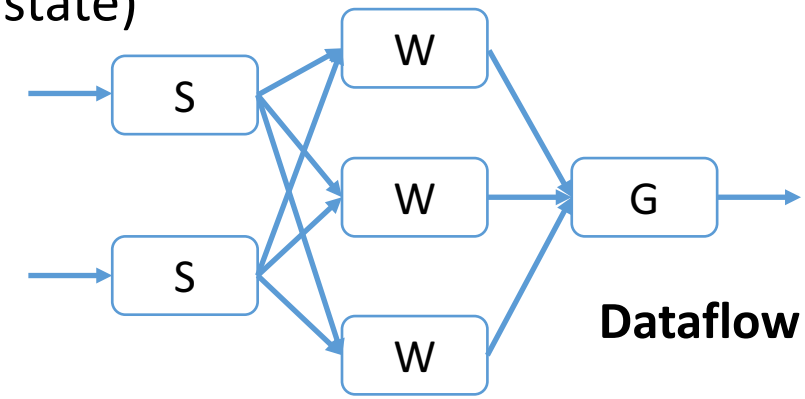
Scheduling Choices

- **Scheduling** is one key area where dataflow systems differ
  - Dynamic Scheduling (Spark)
    - Fine grain control of dataflow graph
    - Graph cannot be optimized
  - Static Scheduling (Flink)
    - Less control of the dataflow graph
    - Graph can be optimized

- Twister2 will allow either

Communication Models

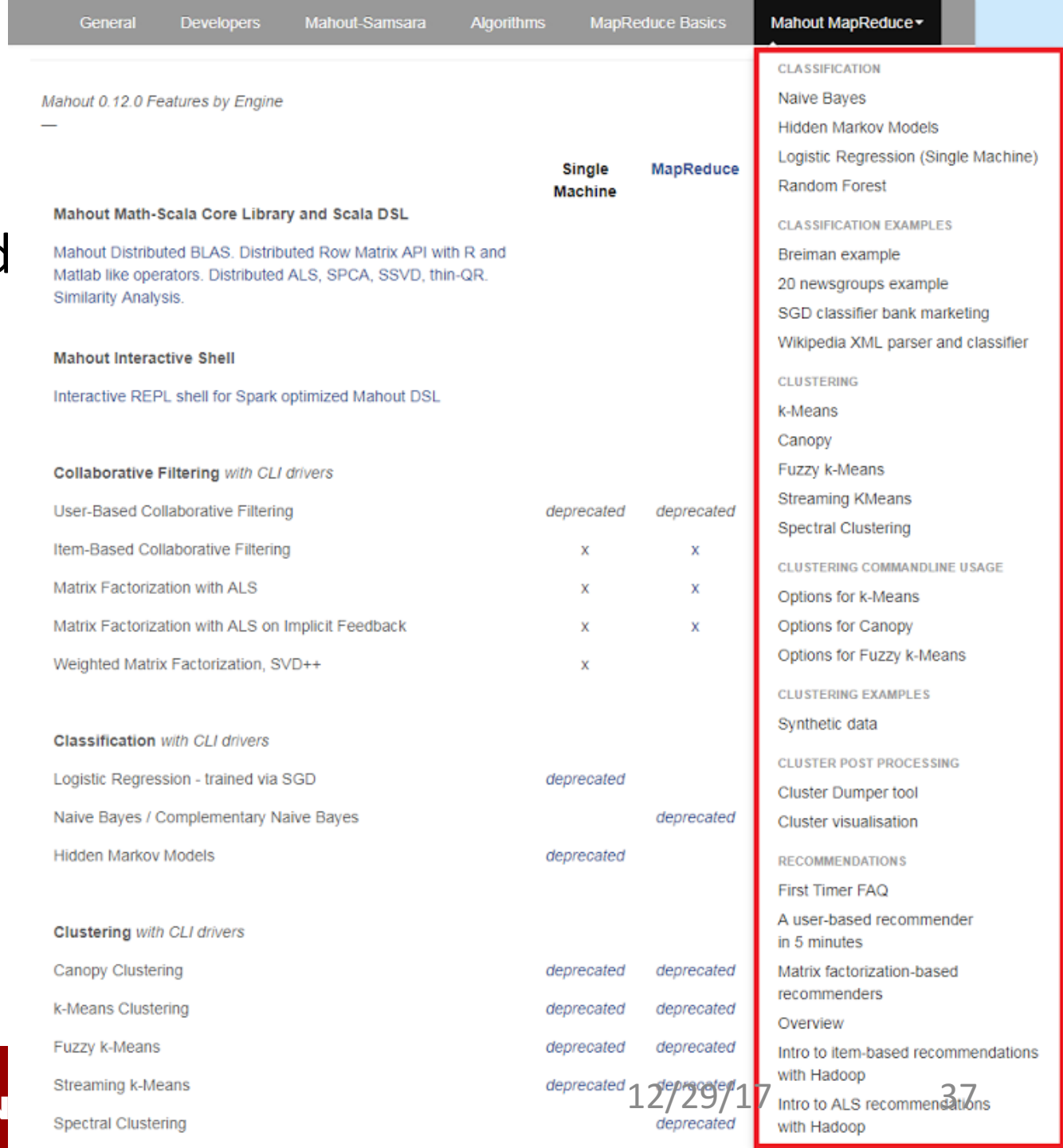
- **MPI Characteristics:** Tightly synchronized applications
  - Efficient communications (µs latency) with use of advanced hardware
  - In place communications and computations (Process scope for state)
- **Basic dataflow:** Model a computation as a graph
  - Nodes do computations with Task as computations and edges are asynchronous communications
  - A computation is activated when its input data dependencies are satisfied



- **Streaming dataflow: Pub-Sub** with data partitioned into streams
  - Streams are unbounded, ordered data tuples
  - Order of events important and group data into time windows
- **Machine Learning dataflow:** Iterative computations and keep track of state
  - There is both Model and Data, but only communicate the model
  - Collective communication operations such as AllReduce AllGather (no differential operators in Big Data problems)
  - Can use in-place MPI style communication

Mahout and SPIDAL

- Mahout was Hadoop machine learning library but largely abandoned as Spark outperformed Hadoop
- SPIDAL outperforms Spark Mllib and Flink due to better communication and in-place dataflow.
- SPIDAL also has community algorithms
  - Biomolecular Simulation
  - Graphs for Network Science
  - Image processing for pathology and polar science



Qiu/Fox Core SPIDAL Parallel HPC Library with Collective Used

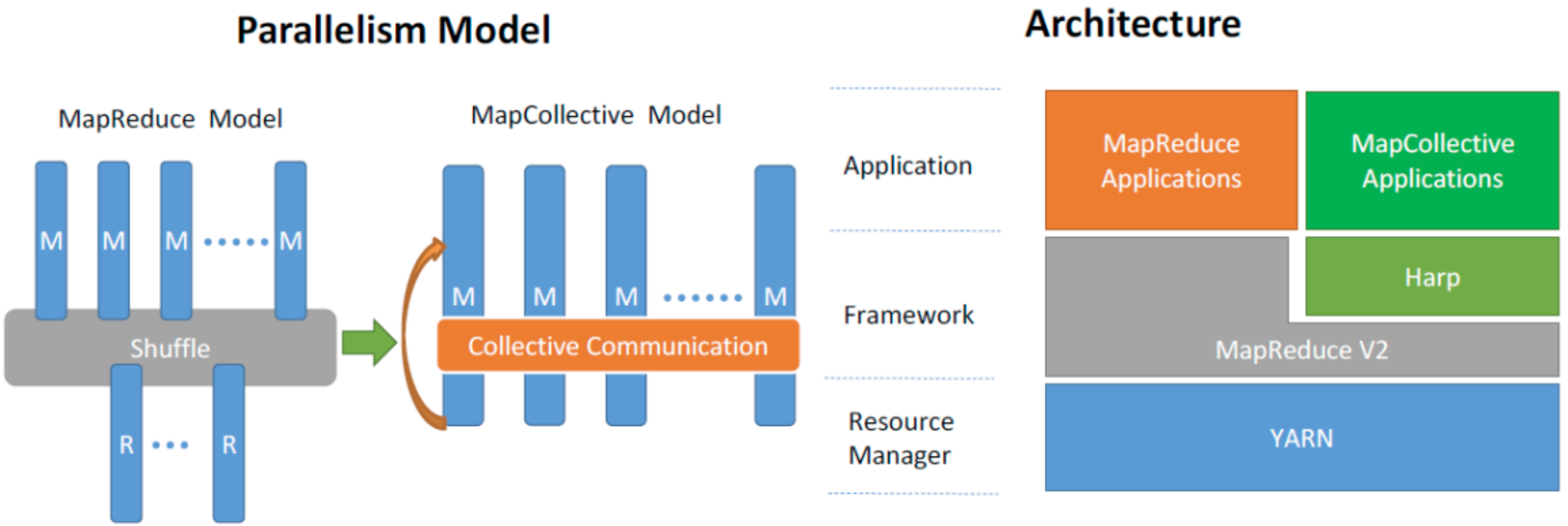
- **DA-MDS** Rotate, AllReduce, Broadcast
- **Directed Force Dimension Reduction** AllGather, Allreduce
- **Irregular DAVS Clustering** Partial Rotate, AllReduce, Broadcast
- **DA Semimetric Clustering (Deterministic Annealing)** Rotate, AllReduce, Broadcast
- **K-means** AllReduce, Broadcast, AllGather DAAL
- **SVM** AllReduce, AllGather
- **SubGraph Mining** AllGather, AllReduce
- **Latent Dirichlet Allocation** Rotate, AllReduce
- **Matrix Factorization (SGD)** Rotate DAAL
- **Recommender System (ALS)** Rotate DAAL
- **Singular Value Decomposition (SVD)** AllGather DAAL

- **QR Decomposition (QR)** Reduce, Broadcast DAAL
- **Neural Network** AllReduce DAAL
- **Covariance** AllReduce DAAL
- **Low Order Moments** Reduce DAAL
- **Naive Bayes** Reduce DAAL
- **Linear Regression** Reduce DAAL
- **Ridge Regression** Reduce DAAL
- **Multi-class Logistic Regression** Regroup, Rotate, AllGather
- **Random Forest** AllReduce
- **Principal Component Analysis (PCA)** AllReduce DAAL

DAAL implies integrated on node with Intel DAAL Optimized Data Analytics Library (Runs on KNL!)

Harp Plugin for Hadoop: Important part of Twister2

Work of Judy Qiu

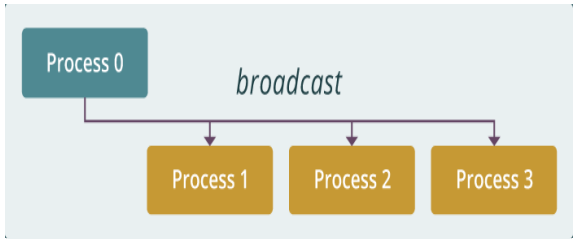


Harp is an open-source project developed at Indiana University [6], it has:

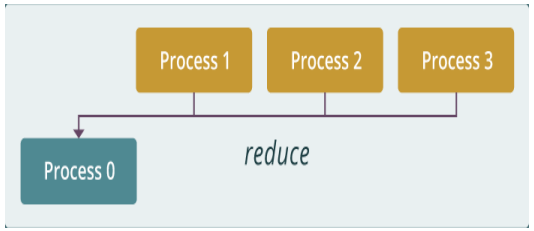
- MPI-like **collective communication** operations that are highly optimized for big data problems.
- Harp has efficient and innovative **computation models** for different machine learning problems.

[6] Harp project. Available at https://dsc-spidal.github.io/harp

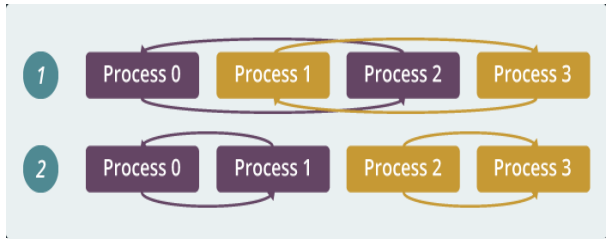
Qiu MIDAS run time software for Harp



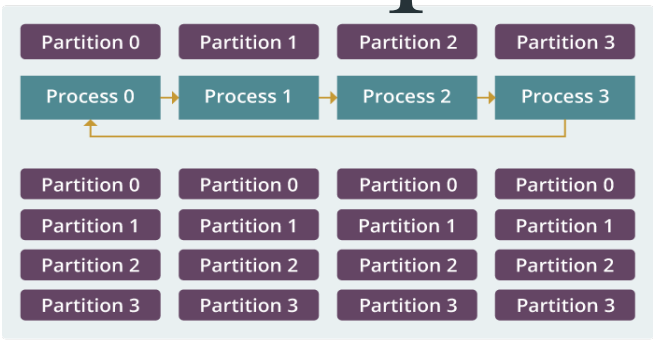
broadcast



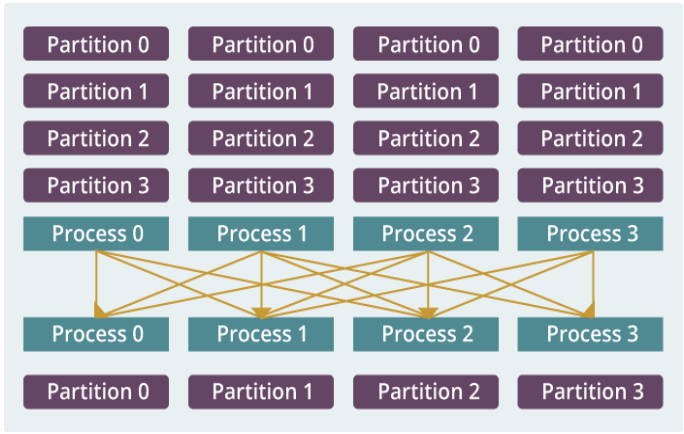
reduce



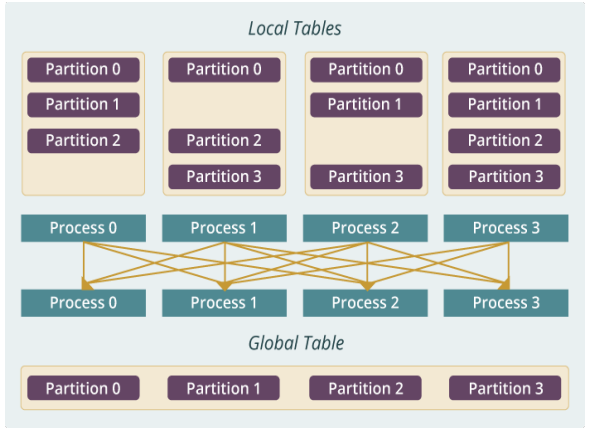
allreduce

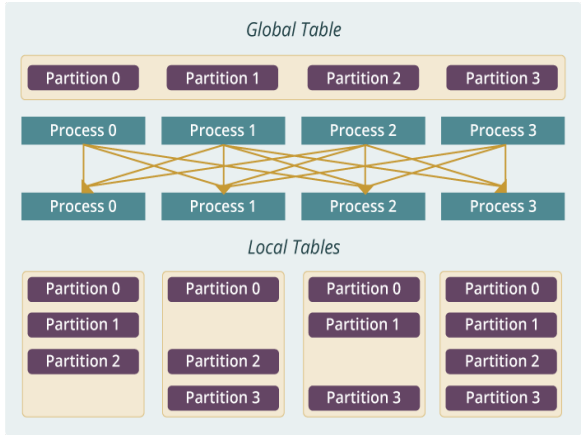


allgather

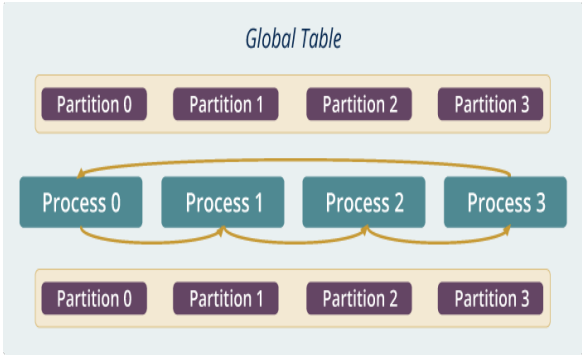


regroup





push & pull



rotate

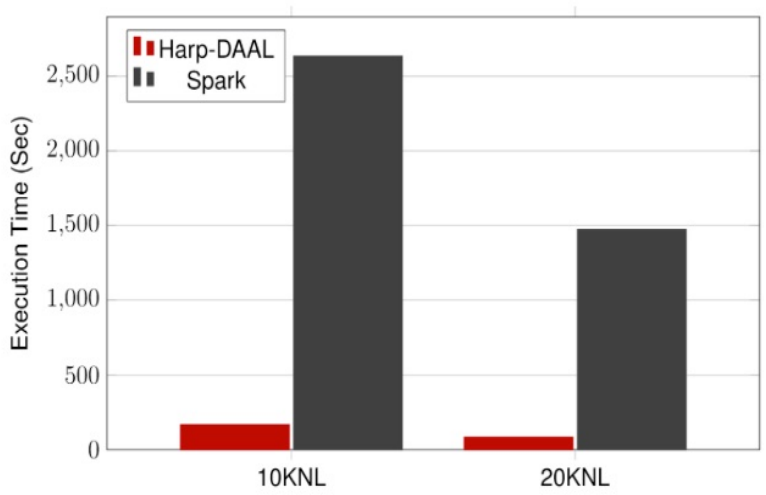
Map Collective Run time merges MapReduce and HPC

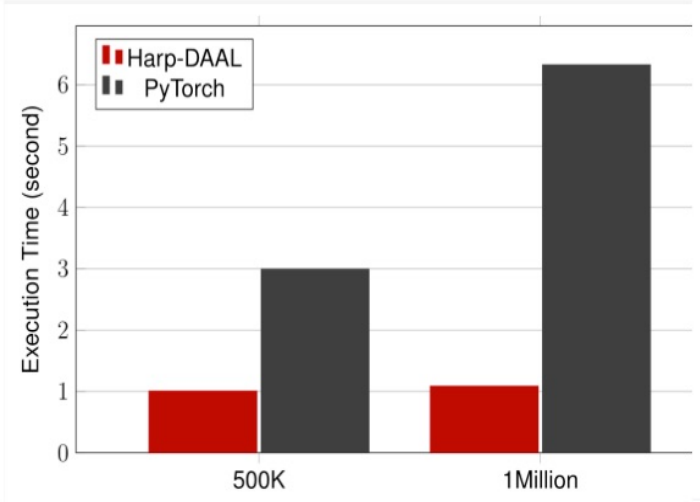
Harp v. Spark

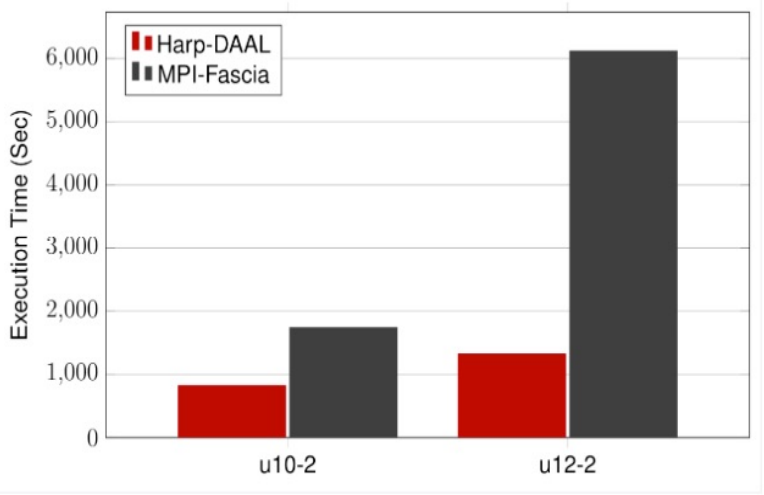
Harp v. Torch

Harp v. MPI

Performance Comparison







K means

- Datasets: 5 million points, 10 thousand centroids, 10 feature dimensions
- 10 to 20 nodes of Intel KNL7250 processors
- Harp-DAAL has 15x speedups over Spark MLlib

PCA

- Datasets: 500K or 1 million data points of feature dimension 300
- Running on single KNL 7250 (Harp-DAAL) vs. single K80 GPU (PyTorch)
- Harp-DAAL achieves 3x to 6x speedups

Subgraph

- Datasets: Twitter with 44 million vertices, 2 billion edges, subgraph templates of 10 to 12 vertices
- 25 nodes of Intel Xeon E5 2670
- Harp-DAAL has 2x to 5x speedups over state-of-the-art MPI-Fascia solution

Implementing Twister2 in detail II

State

Systems State

- **State** is handled differently in systems
  - CORBA, AMT, MPI and Storm/Heron have long running tasks that preserve state
  - Spark and Flink preserve datasets across dataflow node using in-memory databases
  - All systems agree on coarse grain dataflow; only keep state by exchanging data

Spark Kmeans Dataflow

- P = loadPoints()
- C = loadInitCenters()

Iterate

- for (int i = 0; i < 10; i++) {
- T = P.map().withBroadcast(C)
- C = T.reduce() }

Save State at Coordination Point
Store C in RDD

Fault Tolerance and State

- Similar form of **check-pointing** mechanism is used already in HPC and Big Data
  - although HPC informal as doesn't typically specify as a dataflow graph
  - Flink and Spark do better than MPI due to use of **database** technologies; MPI is a bit harder due to richer state but there is an obvious integrated model using RDD type snapshots of MPI style jobs
- Checkpoint **after each stage of the dataflow graph (at location of intelligent dataflow nodes)**
  - Natural synchronization point
  - Let's allows user to choose when to checkpoint (not every stage)
  - Save state as user specifies; Spark just saves Model state which is insufficient for complex algorithms

Initial Twister2 Performance

- Eventually test lots of choices of task managers and communication models; threads versus processes; languages etc.
- Here 16 Haswell nodes each with 1 process running 20 tasks as threads; Java

- Reduce microbenchmark for **Apache Flink** and **Twister2**; Flink poor performance due to nonoptimized reduce operation
- Twister2 has a new dataflow communication library based on MPI – in this case a 1000 times faster than Flnk

Summary of Twister2: Next Generation HPC Cloud + Edge + Grid

- We suggest an event driven computing model built around Cloud and HPC and spanning batch, streaming, and edge applications
  - Highly parallel on cloud; possibly sequential at the edge
- Integrate current technology of FaaS (Function as a Service) and server-hidden (serverless) computing with HPC and Apache batch/streaming systems
- We have built a high performance data analysis library SPIDAL
- We have integrated HPC into many Apache systems with HPC-ABDS
- We have done a very preliminary analysis of the different runtimes of Hadoop, Spark, Flink, Storm, Heron, Naiad, DARMA (HPC Asynchronous Many Task)
- There are different technologies for different circumstances but can be unified by high level abstractions such as communication collectives
  - Obviously MPI best for parallel computing (by definition)
- Apache systems use dataflow communication which is natural for distributed systems but inevitably slow for classic parallel computing
  - No standard dataflow library (why?). **Add Dataflow primitives in MPI-4?**
- MPI could adopt some of tools of Big Data as in Coordination Points (dataflow nodes), State management with RDD (datasets)